Engineer

CLDR

Unicodes data om varje språks datum, tal och sortering

Vad det är

CLDR är en samling data om språk och regioner som kompletterar Unicode. Unicode definierar tecknen som en text skrivs med; CLDR beskriver hur läsare av ett visst språk i en viss region förväntar sig att ett datum, ett tal, ett pris, en enhet eller en lista ska skrivas, och i vilken ordning de förväntar sig att namn sorteras. Unicode-konsortiet förvaltar båda. CLDR:s fullständiga namn är i dag Unicode Common Locale Data & Rules. Tidigare hette det Common Locale Data Repository, och specifikationen hette Locale Data Markup Language, förkortat LDML; enligt specifikationen gäller hänvisningar under de gamla namnen fortfarande.

Projektets webbplats nämner tre delar. Den första är själva datan, utvald och strukturerad för att programvara ska kunna använda den. Den andra är specifikationen, Unicode Technical Standard #35, som beskriver hur datan är uppbyggd och hur algoritmer använder den, med krav på överensstämmelse. Den tredje är koden som samlar in datan från språkspecialister, kontrollerar att den är giltig och konsekvent och konverterar den till de format som utvecklare använder. Specifikationens egen sammanfattning av vad datan täcker är en bra karta: identifierare för locales, måttenheter, talsystem och kalendrar; formatering av datum, klockslag, tidszoner, tal, valutor och mätvärden; samt sortering och sökning. Pluralregler, listmönster och namn på språk och regioner hör också dit.

Datan är ordnad efter locale: ett språk, som kan avgränsas närmare med ett skriftsystem och en region. Varje locale innehåller bara det som skiljer den från dess överordnade locale. Specifikationens exempel är engelska: `en` innehåller huvuddelen av datan, och `en_IE`, engelska som den används på Irland, innehåller bara några få saker, till exempel valutan. Överst, ovanför alla språk, finns en locale som heter root, och localen för ett språks primära region är tom, eftersom den överordnade localen redan innehåller allt den behöver. Några få

uppgifter hör till regionen ensam, oavsett språk: valutan, måttsystemet och konventionerna för veckan. Finns det ingen egen data för franska som den används i USA, får en förfrågan om just den kombinationen därför formaten från `fr` och beloppen i amerikanska dollar.

Källdatan publiceras för varje version som XML i det format som specifikationen definierar, med verktyg som konverterar den till JSON- och POSIX-format. Utgåvorna följer en plan med två cykler per år, och CLDR 49 kom ut den 22 oktober 2026. Ny data kommer in via ett webbaserat verktyg, Survey Tool, som vanligtvis är öppet en gång om året för att samla in data till nya strukturer och rätta data som redan har getts ut, och modersmålstalare och språkexperter granskar det som kommer in.

Använd begreppet i ett beslut

Använd ”Vad är CLDR?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Mjukvara som känns naturlig på varje marknad du säljer på, inklusive datum, valutor och hur namn sorteras.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Byggde ett i18n-system för engelska och danska — 12,027 meddelanden per språkversion, på engelska och danska

Byggde ett i18n-system för engelska och danska Hör av er →

Varför det spelar roll

Projektets webbplats beskriver problemet rakt på sak: att formatera datum, tal, valutor och måttenheter för olika språk och regioner är långt mer komplicerat än de flesta inser. CLDR finns för att inte varje team ska behöva lära sig den komplexiteten ett språk i taget. Projektet stöder över 100 språk, och datan granskas av modersmålstalare och språkexperter, en grundligare kontroll än något produktteam kan ge datumformatet på en marknad det just har gått in på.

De flesta team rör aldrig CLDR direkt, och det är meningen. Projektet räknar alla moderna operativsystem och webbläsare till dem som använder datan, tillsammans med språk som Java, C#, .NET, Swift och JavaScript. Användningen är ofta indirekt: en applikation ber en tjänst i operativsystemet att formatera ett datum, tjänsten anropar ICU, Unicodes produktionsbibliotek för C, C++, Java och Rust, och ICU läser CLDR. Node.js bygger sina `Intl` -objekt på ICU och levererar som standard den fullständiga ICU-datan; Python-program når samma data via Babel och Ruby-program via TwitterCLDR. När ett team anropar en

formateringsfunktion som tar hänsyn till locale kommer svaret nästan alltid från CLDR. Det teamet fortfarande avgör själv är vilken locale det ber om och vilken version det kör.

Alternativet är en tabell skriven för hand: tolv månadsnamn per språk, ett decimaltecken, en tusentalsavgränsare, en ordning för alfabetet. I bästa fall stämmer en sådan tabell den dag den skrivs, och sedan står den still, medan CLDR:s Survey Tool öppnar varje år för att rätta data som redan har getts ut. Den är dessutom oftast skriven av någon som bara talar några av språken i den. Hur danska sorterar de sex ortnamnen nedan skulle en svensk utvecklare knappast gissa, och tvärtom.

Så ser det ut i praktiken

Fråga körmiljön vilken CLDR-version den har med sig, och låt sedan datan sortera sex danska och svenska ortnamn, foga ihop tre danska stadsnamn till en lista och slå upp veckans första dag i fyra locales:

```
console.log("CLDR", process.versions.cldr);

const towns = ["Ørsted", "Odense", "Åsa", "Aabenraa", "Ærø", "Ängelholm"];
console.log("code point", [...towns].sort().join(", "));
for (const tag of ["en", "da", "sv"]) {
  const sorted = [...towns].sort(new Intl.Collator(tag).compare);
  console.log(tag.padEnd(10), sorted.join(", "));
}

const three = ["Aarhus", "Odense", "Aalborg"];
for (const tag of ["en-GB", "en-US", "da", "uk"]) {
  console.log(tag.padEnd(10), new Intl.ListFormat(tag).format(three));
}

for (const tag of ["en", "en-GB", "en-DK", "da"]) {
  const locale = new Intl.Locale(tag);
  const { firstDay } = locale.getWeekInfo();
  console.log(tag.padEnd(10), locale.maximize().toString(), firstDay);
}
```

Node 25.2.1 skriver ut:

```
CLDR 47.0
code point Aabenraa, Odense, Ängelholm, Åsa, Ærø, Ørsted
en         Aabenraa, Ærø, Ängelholm, Åsa, Odense, Ørsted
```

```
da          Odense, Ängelholm, Ærø, Ørsted, Aabenraa, Åsa
sv          Aabenraa, Odense, Åsa, Ängelholm, Ærø, Ørsted
en-GB       Aarhus, Odense and Aalborg
en-US       Aarhus, Odense, and Aalborg
da          Aarhus, Odense og Aalborg
uk          Aarhus, Odense i Aalborg
en          en-Latn-US 7
en-GB       en-Latn-GB 1
en-DK       en-Latn-DK 1
da          da-Latn-DK 1
```

Första raden är den version som körmiljön är byggd med. Den är 47.0, trots att CLDR 49 har kommit ut: en körmiljö har med sig datan från sitt eget bygge, inte den nyaste data som finns.

En vanlig `sort()` utan jämförelsefunktion lägger Ä, Å, Æ och Ø efter alla bokstäver från A till Z, i ordning efter deras teckenkoder, och det stämmer inte med något av de tre språken. Engelska sorterar varje namn efter dess grundbokstav, så de fyra namnen som börjar med A kommer först och Ørsted kommer efter Odense. Danska avslutar alfabetet med Æ, Ø och Å, behandlar Ä som en variant av Æ och sorterar Aabenraa, stavat med dubbel-a enligt äldre bruk, bland Å-namnen; specifikationens del om sortering kallar detta den danska kontraktionen ”aa”. Svenska avslutar alfabetet med Å, Ä och Ö, behandlar Æ som en variant av Ä och Ø som en variant av Ö, och sorterar Aabenraa som ett namn som börjar med två a. Norskt bokmål, `nb`, ger samma ordning som danska. Inget av detta står i koden; varje ordning kommer från CLDR:s sorteringsregler för språket i fråga.

Listorna fungerar på samma sätt. Varje locale har ett listmönster med egna delar för början, mitten och slutet av en lista och för en lista med två element. Brittisk engelska sätter inget kommatecken före ”and”, amerikansk engelska gör det, danska binder ihop med ”og” och ukrainska med ”i”, och allt kommer från ett anrop som inte innehåller ett enda ord på något av de språken.

De fyra sista raderna visar data som hör till en region och inte till ett språk. Varje rad ger språkkoden, den fullständiga form som `maximize()` fyller i med CLDR:s mest sannolika skriftsystem och region, och veckans första dag som ett tal enligt ISO 8601, där 1 är måndag och 7 är söndag. Körmiljön tolkar ett `en` utan region som engelska med latinsk skrift, så som språket används i USA, och därför börjar veckan på söndag. För engelska i Storbritannien eller i Danmark börjar veckan på måndag, precis som för danska: veckan kommer från regionen, oavsett språk.

Konventionerna som CLDR registrerar är också de som en översatt text måste få rätt för hand. Ett av projekten på den här webbplatsen, en sifferkontroll mellan språken, plockar ut varje tal, varje procentsats, varje multiplikator och varje enhet ur en engelsk sträng och kräver att vart och ett finns med i varje översättning. Den kontrollerar också varje språk mot de konventioner det måste använda och inte får använda, bland annat decimaltecken, tusentalsgruppering och citattecken.

Var det går fel

Formaterad text jämförs som en fast sträng. Ett test som förväntar sig `"1 500"` av `new Intl.NumberFormat("sv").format(1500)` misslyckas trots att resultatet ser ut exakt så, eftersom svenska, norska och ukrainska skiljer tusental åt med U+00A0 NO-BREAK SPACE, som en editor visar som ett vanligt mellanslag. Samma test kan också gå sönder vid en uppgradering, eftersom en ny CLDR-version kan rätta data som en äldre version levererade. Jämför formaterad text med det som samma formaterare ger, kontrollera strukturen med `formatToParts`, som märker varje del som integer, group, decimal eller fraction, och tolka aldrig formaterad text tillbaka till ett tal. Specifikationens råd är att lagra och utbyta data i språkneutral form och formatera den för läsaren.

Server och webbläsare har var sin version. Node 25.2.1 levererar CLDR 47.0 medan projektet har kommit till 49, och en webbläsare har med sig den version den själv byggdes med. En sida som formaterar ett datum på servern och sedan igen i webbläsaren kan visa två olika strängar för ett och samma värde, och en ögonblicksbild som har tagits på en dator kan misslyckas på en annan. Formatera varje värde på ett ställe, spara `process.versions.cldr` bredvid testernas ögonblicksbilder och uppgradera körmiljön medvetet.

En locale som saknas faller tillbaka utan fel. `new Intl.NumberFormat("tlh")` formaterar som amerikansk engelska utan att kasta ett fel, eftersom körmiljön saknar data för klingonska och faller tillbaka till sin förvalda locale. Dokumentationen för Node.js beskriver ett lömskare fall: en binärfil byggd med alternativet small-icu har vanligtvis bara engelsk data, och där skrivs ett spanskt månadsnamn ut som ”M01” eller ”January” i stället för ”enero”. Kontrollera `resolvedOptions().locale` eller `supportedLocalesOf` innan du litar på en locale, och bygg med den fullständiga ICU-datan, som är standard i Node och det som de officiella binärfilerna använder.

En språkkod utan region får den mest sannolika. Körmiljön tolkar `en` som `en-Latn-US`, så en läsare i Irland, Storbritannien eller Danmark som serveras `en` ser en vecka som börjar på söndag och datum som skrivs 10/27/2026, med månaden först. Datan bakom, som specifikationen kallar likely subtags, är enligt dess egen beskrivning heuristiskt härledd och kan ändras med tiden. Skicka med läsarens region tillsammans med språket när den är känd, och anropa `maximize()` när bara språket är känt, så att du ser vilken region körmiljön kommer att utgå från.

CLDR:s identifierare skickas till ett BCP 47-API. CLDR:s filer och exempel skriver locales med understreck, till exempel `en_IE`, och specifikationen behandlar `_` och `-` som likvärdiga. Det gör inte JavaScripts `Intl`: `new Intl.NumberFormat("da_DK")` kastar `RangeError: Invalid language tag: da_DK`. Konvertera vid gränsen på samma sätt som specifikationens egen konvertering till BCP 47, genom att byta varje `_` mot `-`, och skicka resultatet genom `Intl.getCanonicalLocales`, som returnerar `da-DK`.

Läs mer

- Unicode CLDR Project — vad CLDR är, de tre delarna, vem som använder det och hur, och utgivningsplanen.
- UTS #35, Part 1: Core — specifikationen: locale-identifierare, de tidigare namnen, arv mellan locales, likely subtags och rådet att hålla data språkneutral.
- UTS #35, Part 2: General — visningsnamn, måttsystem och de listmönster som används ovan.
- UTS #35, Part 5: Collation — hur CLDR anpassar standardsorteringen till varje språk, med danska som exempel.
- Node.js: Internationalization support — hur Node bygger sina `Intl`-objekt på ICU, byggalternativen och hur man tar reda på vad en binärfil stöder.
- ECMAScript Internationalization API Specification — standarden bakom `Intl`, bland annat de `Locale`-metoder som används ovan och CLDR:s veckodata och likely subtags som de bygger på.

Ordlista

Välj utifrån era faktiska förutsättningar

Beskriv arbetsflödet, befintliga gränssnitt och användarna som behöver förändringen. Vi kan definiera en testbar leverans och en överlämning som gör att teamet kan underhålla lösningen.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är CLDR?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

welcome@engineer.company +45 50 10 61 84 Köpenhamn, Danmark

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/cldr/